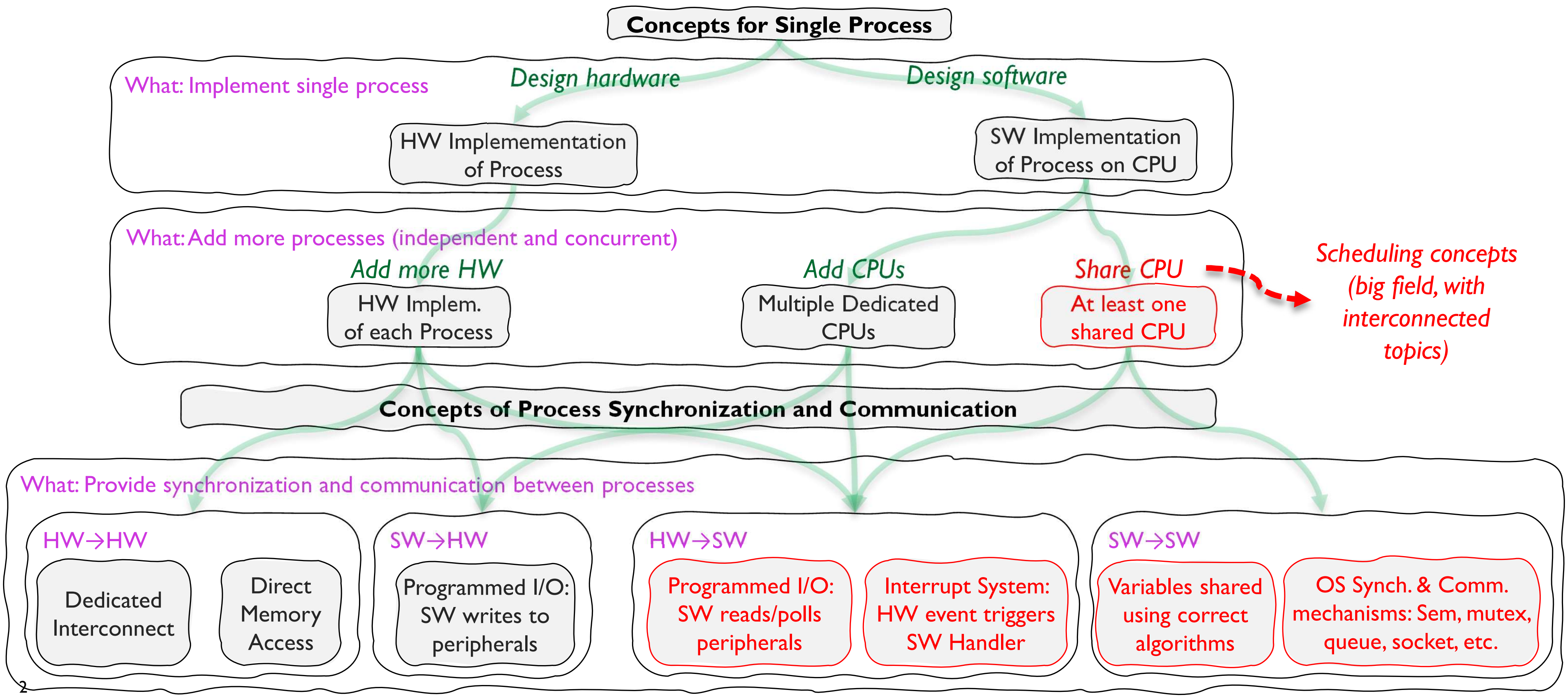
Key Ideas in Embedded System Responsiveness: Concurrent Processes in Software and Hardware

3/4/2025

NC STATE UNIVERSITY

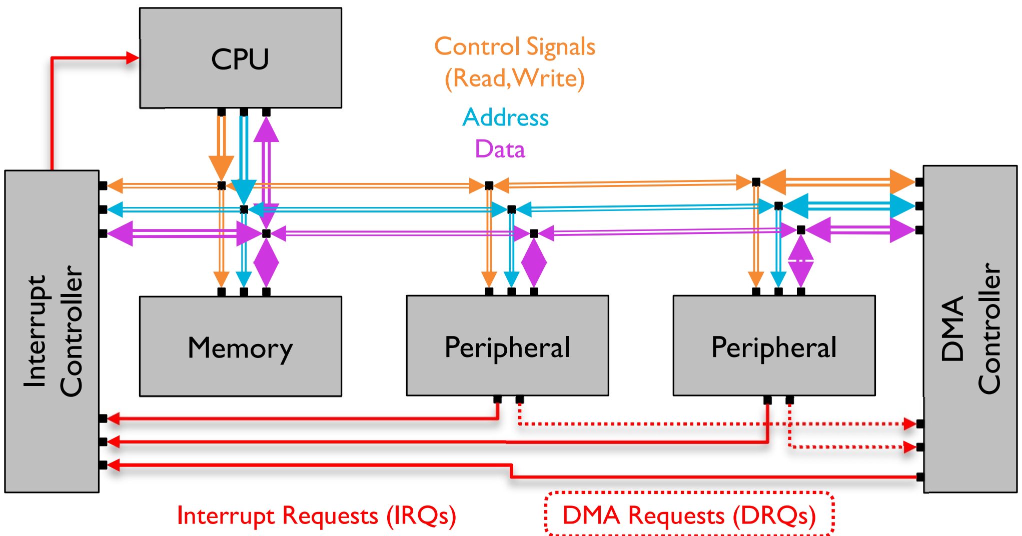
Big Picture 1: Concurrent HW and SW Processes in System

Start simple, then examine options as you build up the system



Refresher: Direct Memory Access Controller

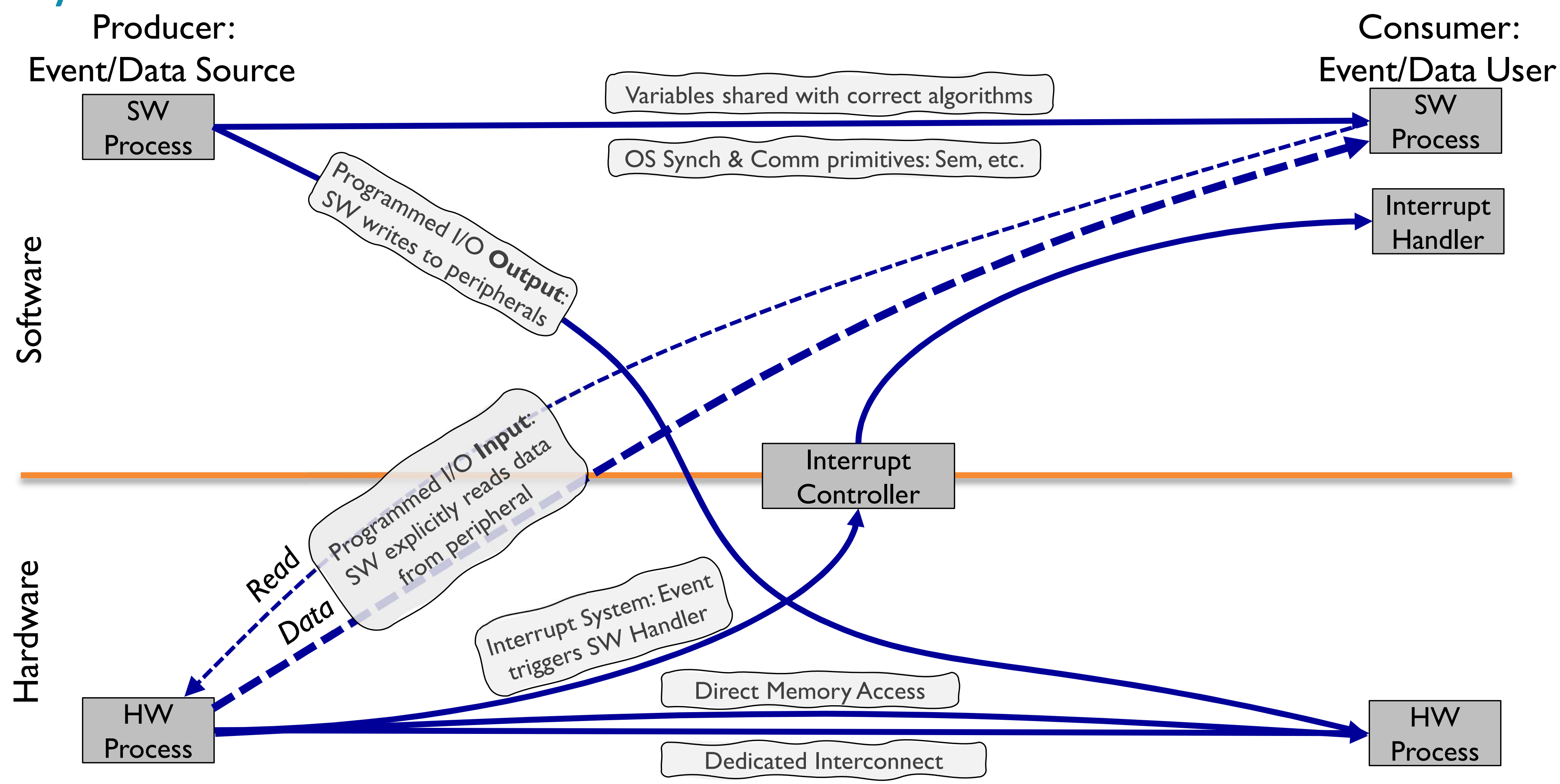
Allows Hardware->Hardware communication without using CPU



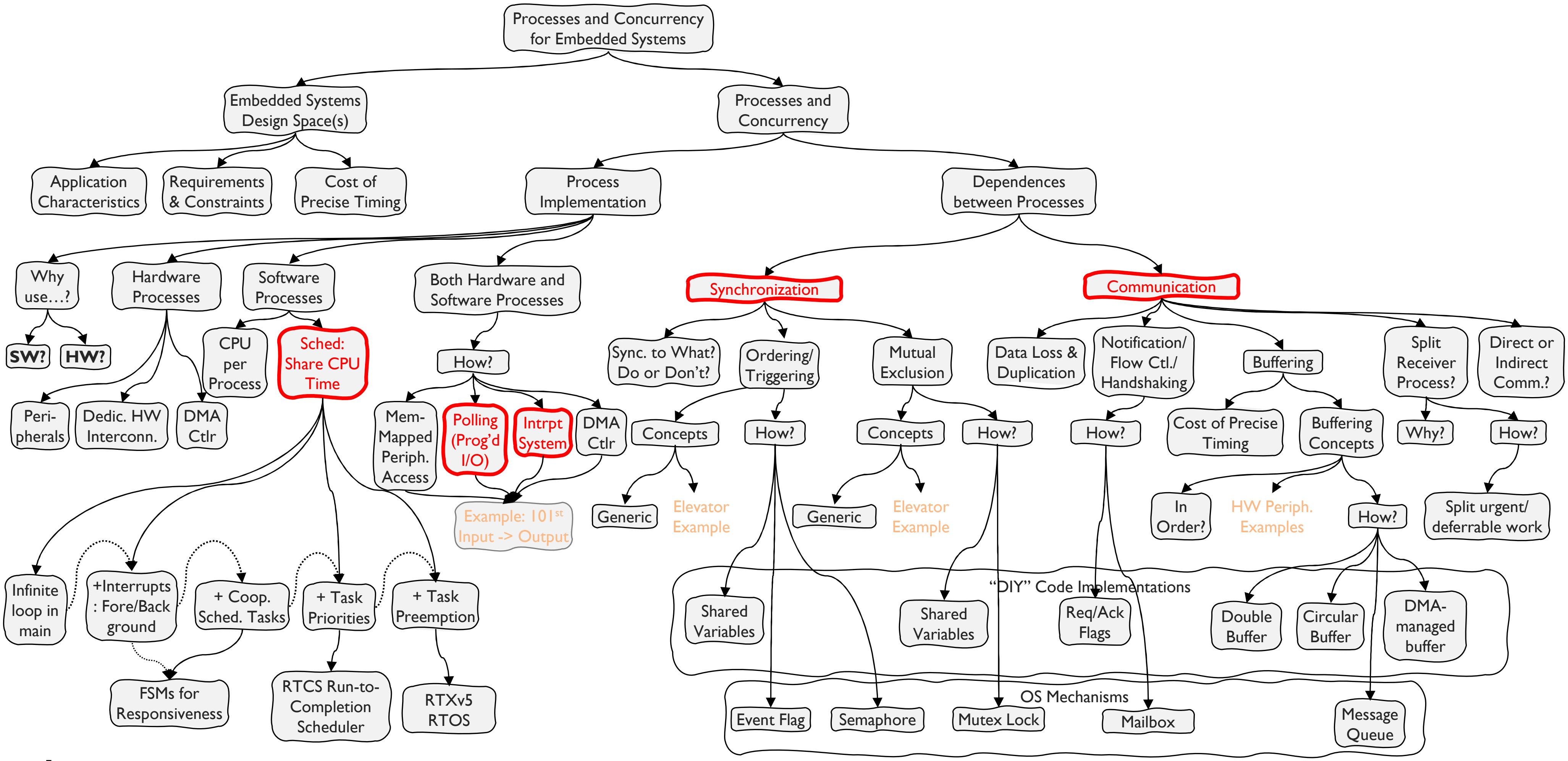
- How to access memory and peripherals?
  - CPU uses **memory bus** (address, data, control) to access memory and peripheral devices
  - Memory bus can also be controlled by DMA Controller (DMAC) peripheral

- DMA features
  - DMAC can transfer (copy) N data items within memory space from SrcAdx to DstAdx
    - SrcAdx, DstAdx: fixed or increment per item copied
    - Allows direct copy, but also accessing sequential items in memory array ("Save the next N ADC data values in memory starting at this address")
  - Transfer can be triggered by:
    - Hardware (DMA Request from peripheral device)
    - Software (CPU writing to DMA request control register)
  - Configurable bus sharing with CPU: can be greedy (burst of all transfers), round-robin, etc.
  - DMAC can generate interrupt when done
  - DMAC has multiple channels, each with individual trigger source, Adx pointers and behaviors, item count, interrupt behavior

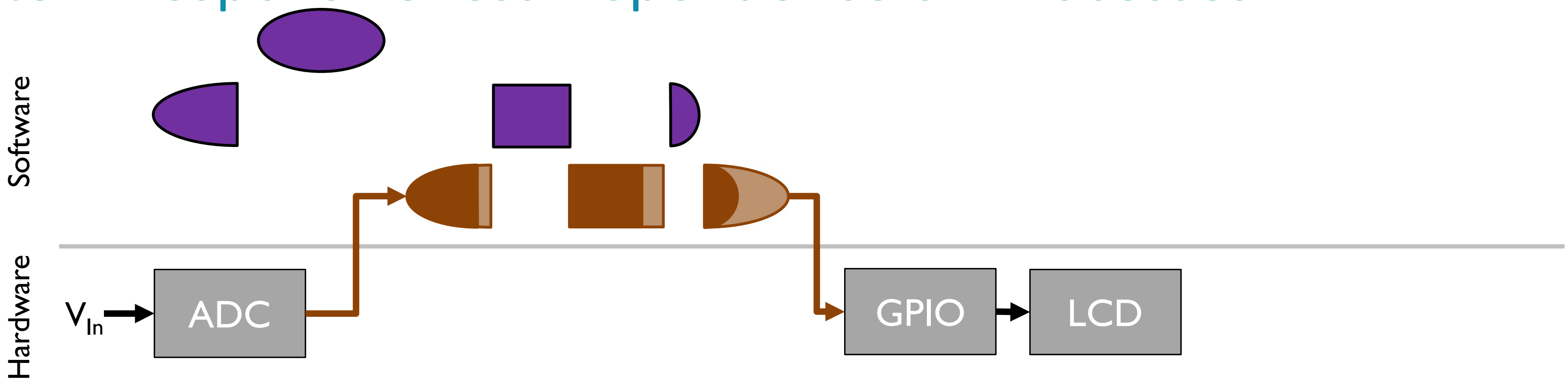
Sync. and Comm. Paths for HW and SW Processes



Big Picture 2: Synchronization, Communication and Scheduling

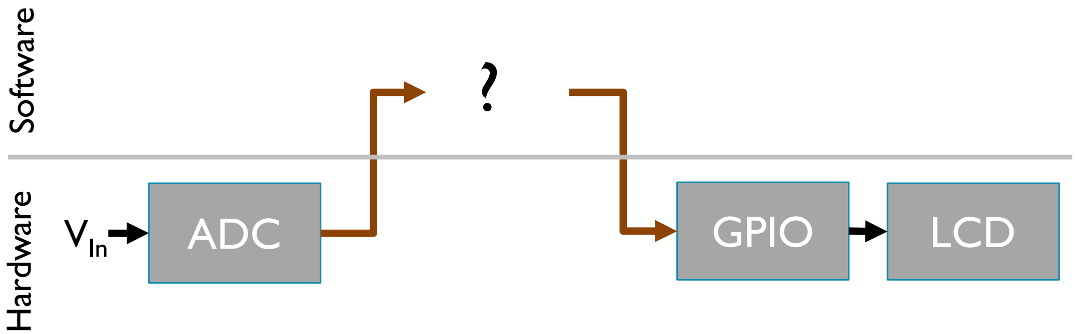


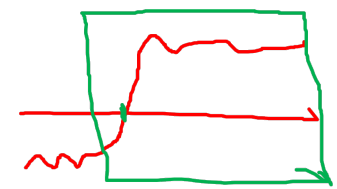
System Responsiveness Dependence on Processes



- System responsiveness depends on **all** activities along a chain of processes
- Timing for hardware processes is fast, very stable
  - Little hardware shared with other processes (e.g memory buses)
- Timing for software processes is much slower and unstable
  - Time to execute a software process may vary if input data triggers different control-flow behavior (conditionals, loops, etc.)
- Sharing CPU among multiple software processes delays a given process due to
  - Timing interference from other processes (preemption, blocking)
  - Inherent delays and processing overhead for:
    - Synchronization: deciding if process may run (is ready) or must wait for event/condition
    - Scheduling: deciding which ready software process to run next
    - Dispatching: starting that software process running

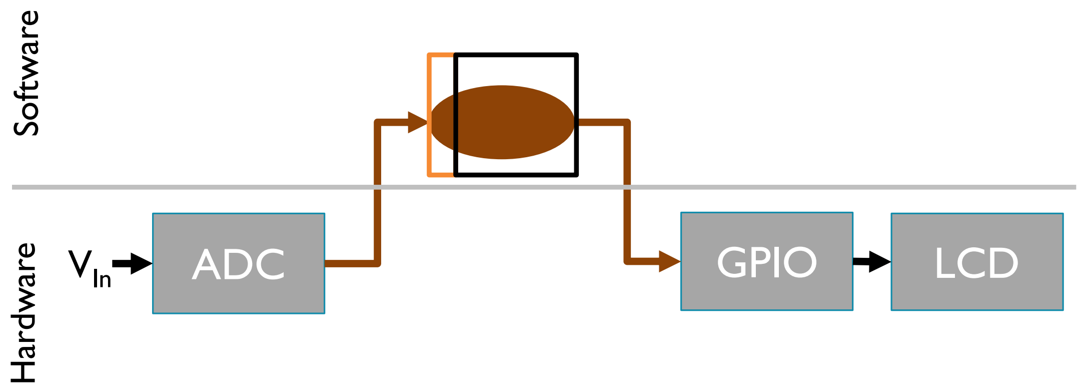
Example System: Oscilloscope with Triggering





- Behavior
  - Display signal voltage on LCD, synchronized to signal's rising edge through threshold
  - Simplify: Ignore erasing previous acquisition from LCD. Ignore user controls
- Hardware Components
  - Analog to Digital converter: Fast enough to keep up with MCU: 48 Msamples/second. (NOT the ADC on the KL25Z)
  - Microcontroller: KL25Z
  - LCD: 320x240 display with controller
- Basic flow of operations
  - **Wait for/detect** V_{in} rising through $V_{threshold}$
    - Rising Edge: V_{in}(previous sample) < $V_{threshold}$ AND V_{in}(current sample) >= $V_{threshold}$
  - Loop to **Acquire** N data samples and **display** them
    - Sample ADC: acquisition[n] = ADC_data
    - Scale acquisition[n], plot on LCD until reaching end of display

Simple Busy-Wait Loop

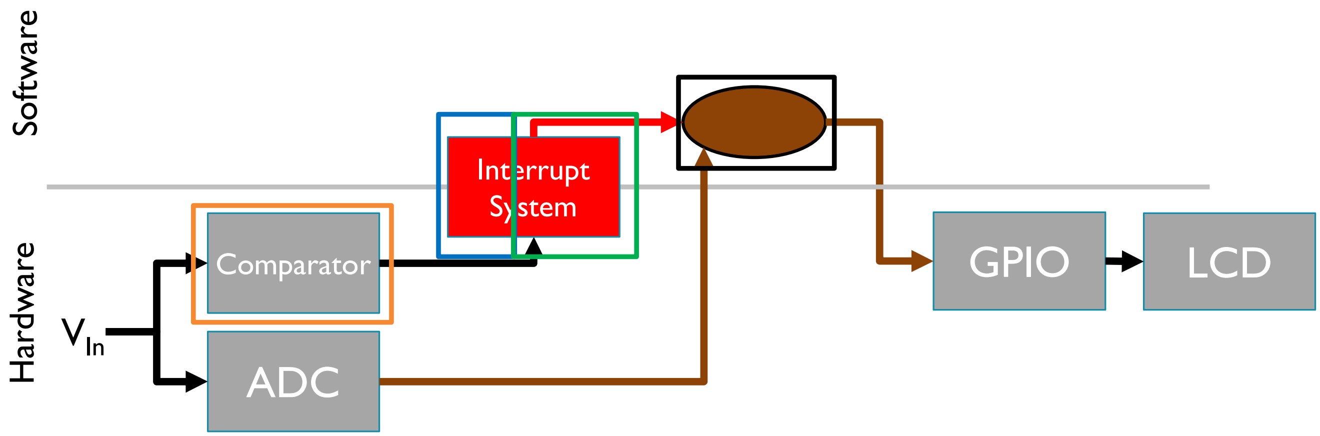


- Synchronize: In Process A
- Schedule: Implicit
- Dispatch: Implicit

Process A

```
…
// Detector/Synchronizer
while (ADC->Result < V_Threshold)
  ;
// No Scheduler
// No Dispatcher
// Handler process
x = 0;
for (n=0; n<NS; n++) {
  r = ADC->Result;
  y = scale(r);
  LCD_Plot(x++,y);
}
```

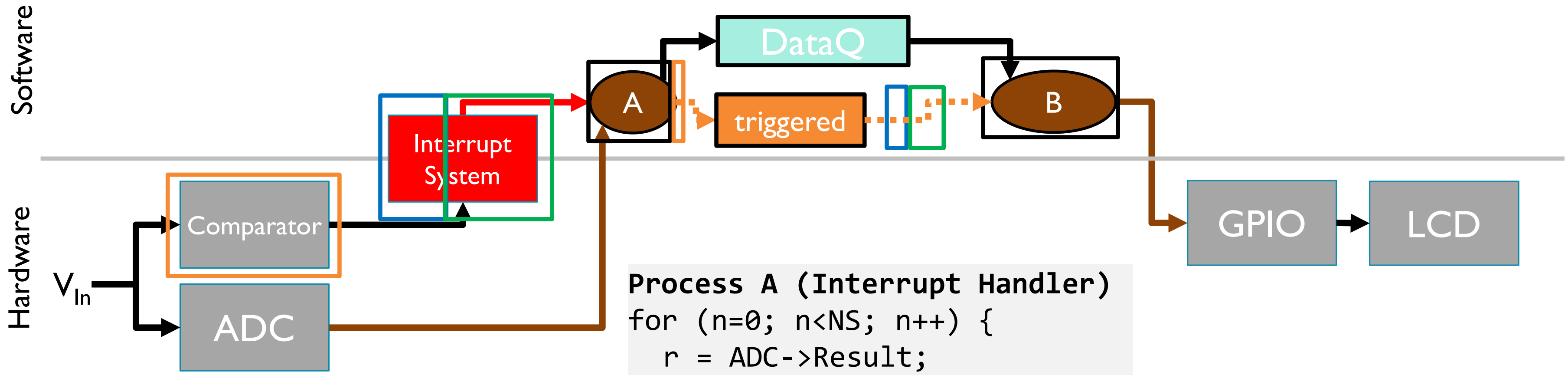
Hardware Help: Comparator Interrupt



- Synchronize: Comparator
- Schedule: Interrupt System
- Dispatch: Interrupt System

```
Process A (Interrupt Handler)
// No Detector/Synchronizer
// No Scheduler
// No Dispatcher
// Handler process
x = 0;
for (n=0; n<NS; n++) {
  r = ADC->Result;
  y = scale(r);
  LCD_Plot(x++,y);
}
```

Hardware & OS Help: Comparator Interrupt, Deferred Plotting



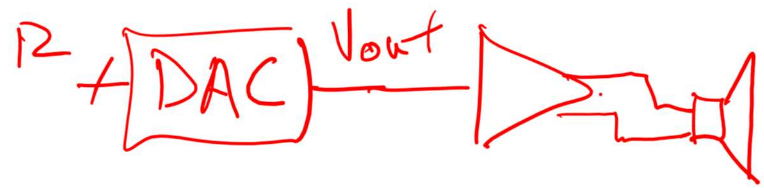

```
Process A (Interrupt Handler)
for (n=0; n<NS; n++) {
  r = ADC->Result;
}
osEventFlagSet(triggered);
```

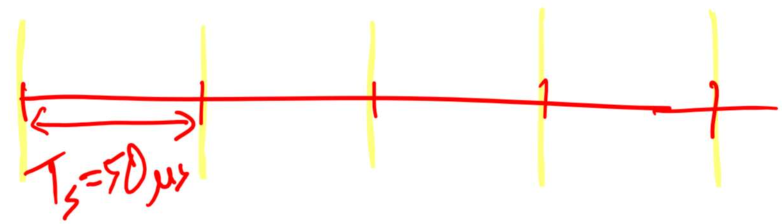
- Acquire Data (Process A)
  - Synchronize: Comparator, Interrupt System
  - Schedule: Interrupt System
  - Dispatch: Interrupt System
- Plot Data (Process B)
  - Synchronize: Process A, OS
  - Schedule: OS
  - Dispatch: OS

```
Process B
while (1) {
   osEventFlagWait(triggered);
   x = 0;
   for (n=0; n<NS; n++) {
    y = scale(DataQ[n]);
    LCD_Plot(x++,y);
   }
}
```

Hardware Help for Stabilizing Timing of Data Output

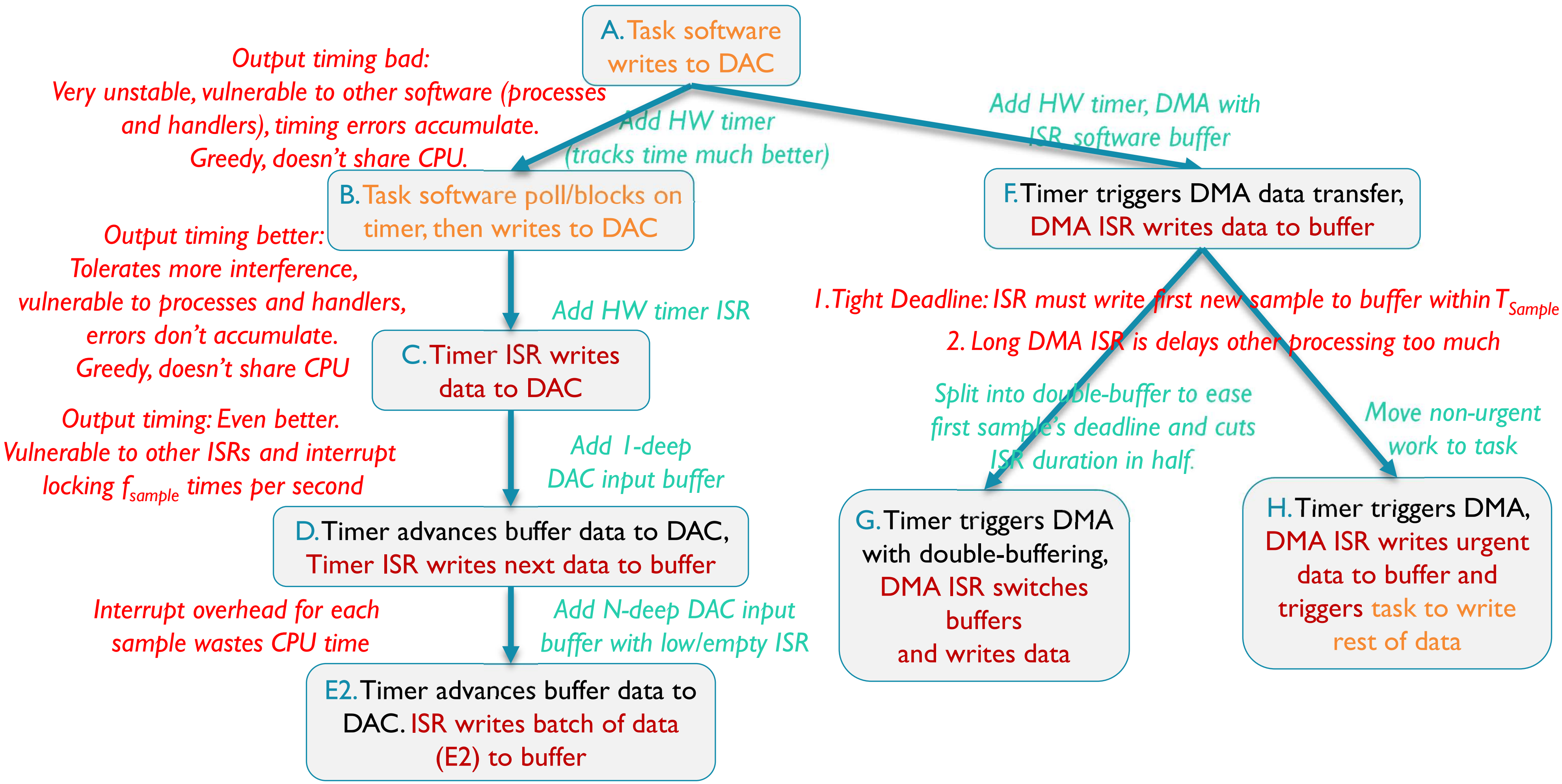
- Want to update DAC output every 50 us for a 20 kHz update rate
  - DAC signal amplified to drive speaker
- Timing analysis approach - Vulnerabilities?
  - What kinds of events and over what time periods can affect the output update time?
- Solutions
  - Use hardware to help (or even replace) software doing synchronization, scheduling, or work.
    - Synchronization: determining when to update output
    - Scheduling: selecting code to run
    - Work: updating output
  - Buffer data to loosen (simplify) software timing requirements
- More details in ECE 460/560 slides





NC STATE UNIVERSITY

Design Evolution Roadmap and Reasoning



Software and Hardware Components as Design Evolves